www.javaspektrum.de

he 6/2019
NF 18,20

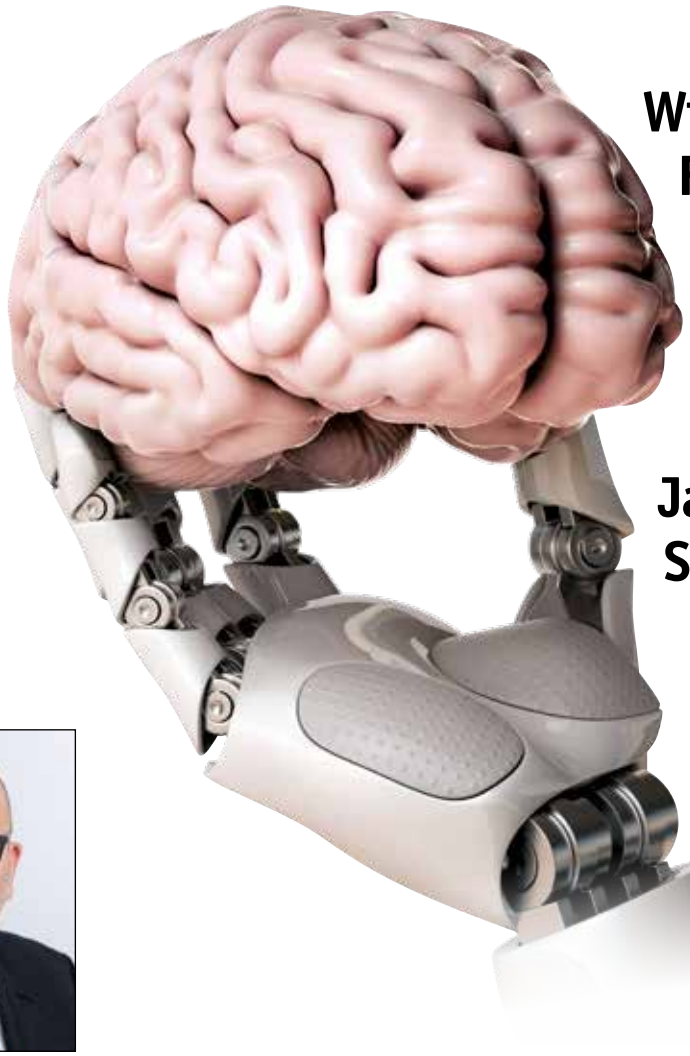
4 194156 109901 06

JavaSPEKTRUM

Magazin für professionelle Entwicklung und Integration von Enterprise-Sy

Mensch2Maschine & Maschine2Maschine

Moderne Schnittstellen



Wie RxJS hilft, Frontends zu entwerfen

Aufbau eines Java-basierten Open-Source-MQTT-Stack

Interview

Frank Benke, Head of IT der Hahn Group über Software Defined Infrastrukturen und warum er nicht auf die Cloud setzt

Fachthemen

Security @ Kubernetes – Network Policies – Teil 2

Plattformen für DevOps-Entwickler: Leistungen und Limits

Mensch2Maschine & Maschine2Maschine Moderne Schnittstellen

G30325

Security @ Kubernetes

Network Policies – Teil 2: Fortgeschrittene Themen und Tipps

Johannes Schnatterer

Beim Deployment von Anwendungen auf managed Kubernetes-Clustern ist der Betrieb für die Sicherheit zuständig, richtig? Nicht ganz! Zwar abstrahiert Kubernetes von der Hardware, seine Programmierschnittstelle bietet Entwicklern dennoch viele Möglichkeiten, die Sicherheit der darauf betriebenen Anwendungen gegenüber der Standardeinstellung zu verbessern. Dieser Artikel thematisiert die fortgeschrittenen Themen zu Network Policies, wie CNI, Test, Debugging, Einschränkungen, Alternative und Fallstricke.

In einem Kubernetes-Cluster können standardmäßig alle (Nodes, Pods, Kubelet usw.) miteinander kommunizieren. Gelingt es einem Angreifer, eine Sicherheitslücke in einer der Anwendungen auszunutzen, kann er seinen Angriff dadurch leicht auf alle dahinter liegende Systeme im gleichen Cluster ausbauen. Durch das Kubernetes-Bordmittel Network Policies kann dies eingeschränkt werden. Der erste Teil dieser Artikelserie empfiehlt Whitelisting des ein- und ausgehenden Datenverkehrs [Sch19]. Wer dies selbst in einer definierten Umgebung ausprobieren will, findet vollständige Beispiele mit Anleitung im Repository „cloudogu/k8s-security-demos" bei GitHub [k8s-sec-demo].

Doch wie sieht dies in der Praxis aus: Welche Fallstricke gibt es beim Thema Network Policies? Wo hat der Mechanismus seine Grenzen? Gibt es Alternativen? Wie kann sichergestellt werden, dass die Network Policies so funktionieren, wie gedacht? Was tun im Fehlerfall? Auf diese fortgeschrittenen Themen gibt dieser Teil Antworten.

CNI-Plug-in-Support

Wie im ersten Teil erwähnt, werden Network Policies zwar in Kubernetes spezifiziert, aber vom Container Networking Interface (CNI)-Plug-in durchgesetzt. Spezifikation und Implementierung können also abweichen. Beispielsweise

- unterstützt das CNI-Plug-in Flannel generell keine Network Policies,
- wurde die Programmierschnittstelle mehrfach erweitert („egress" ab Kubernetes 1.8, „namespaceSelector" und „podSelector" gleichzeitig verwendbar ab 1.11), weshalb diese Features erst in neueren Versionen der CNI-Plug-ins implementiert sind, und
- bei einem systematischen Test verschiedener CNI-Plug-ins zeigte sich, dass WeaveNet viele Network Policies anders auslegt als Calico [test-netpol].

Daher empfiehlt es sich, ein CNI-Plug-in gründlich zu evaluieren, bevor es eingesetzt wird. Hier kann der Kubernetes-Conformance-Test „Sonobuoy" hilfreich sein. Damit kann man testen, ob ein Kubernetes-Cluster in seiner aktuellen Konfiguration alle Features

Johannes Schnatterer ist Continuous-Delivery-Enthusiast, fokussiert auf Softwarequalität und hat eine Passion für Open Source. Er ist begeistert von allen Cloud-native-Themen und deren lebhafter Community. Johannes Schnatterer arbeitet für die Cloudogu GmbH als Solution Architect und Trainer. E-Mail: johannes.schnatterer@cloudogu.com

laut Spezifikation unterstützt. Dies lässt sich einfach für das Feature Network Policies durchführen [netpol-e2e-test].

Nutzer eines „managed" Clusters haben keinen Einfluss auf das verwendete CNI-Plug-in. Sie sind davon abhängig, ob der Anbieter das Feature Network Policies bereitstellt. Typischerweise muss das Feature außerdem für den Cluster explizit aktiviert werden, da sonst die Network Policies nicht durchgesetzt werden. Die großen Anbieter setzen häufig auf das CNI-Plug-in Calico, beispielsweise Googles GKE (verfügbar seit März 2018 [gke-netpol]), Amazons EKS (verfügbar seit Juni 2018 [eks-netpol]) sowie Microsofts AKS (verfügbar seit Mai 2019 [aks-netpol]).

Test und Debugging

Es ist also empfehlenswert, am laufenden Kubernetes-Cluster zu validieren, dass die Network Policies gemäß Spezifikation durchgesetzt werden. Derzeit hat sich noch kein Tool etabliert, mit dem automatisierte Tests ausführbar sind. Für manuelles Testen gibt es jedoch viele Möglichkeiten, die auch gut zum Debugging verwendet werden können.

Im einfachsten Fall öffnet der Nutzer eine Shell im Container per „kubectl exec". Dies funktioniert natürlich nur, wenn das zugehörige Image eine Shell enthält. Sehr wahrscheinlich sind im Container allerdings nicht viele Werkzeuge enthalten, mit denen Netzwerkzugriffe durchführbar sind. Um die Angriffsfläche so klein wie möglich zu halten, ist es gute Praxis, so wenig Pakete wie möglich in Container-Images für die Produktion zu installieren.

Ein einfaches Mittel, um solche minimalen Container zu debuggen, ist das Ausführen temporärer Container, welche die notwendigen Tools enthalten. Zum Thema Netzwerk bietet sich hier das Image „nicolaka/netshoot" an, das viele Netzwerk-Utilities (wie curl, ifconfig, nmap, ngrep, socat, tcpdump usw.) enthält. Technisch kann ein Container in der gleichen „Umgebung" wie ein anderer gestartet werden, indem beide in denselben Linux-Namespaces (PID, Network usw.) ausgeführt werden. Bei Docker ist dies für den Network-Namespace mittels

```
docker run --net container:<container_name> nicolaka/netshoot
```

einfach möglich. Bei Kubernetes ist es herausfordernder. Hier abstrahieren Pods vom Zugriff auf die Container. Alle Container eines Pods teilen sich die Linux-Namespaces. Allerdings ist es mit „kubectl" 1.15 noch nicht ohne Weiteres möglich, kurzlebige neue Container in einem Pod zu starten. Perspektivisch wird der „kubectl debug"-Befehl jedoch den Start von kurzlebigen Containern für das Debugging erlauben [kubectl-debug]. Bis dahin gibt es folgende Möglichkeiten (s. Listing 1 für konkrete Beispiele):

- Temporäre Pods können mit den gleichen Labels wie der eigentliche Pod gestartet werden. Dies ist einfach durchzuführen und beeinträchtigt den eigentlichen Pod nicht. Allerdings befinden sich die Pods dann nicht im gleichen Linux-Namespace. Für viele Network Policy-Testfälle reicht dies jedoch aus.
- Einem Deployment können explizit weitere Container (oft auch „Sidecar" genannt) hinzugefügt werden. Die Umsetzung per YAML ist allerdings umständlich und erfordert ein Neuanlegen des Pods. Daher ist sie für die Produktion nur bedingt geeignet. Der neue Container muss nach dem Debugging wieder manuell entfernt werden, was einen weiteren Neustart bedingt.
- Existiert Zugriff auf die Nodes beziehungsweise Knoten, können direkt über die Container-Runtime weitere Container im Namespace des gewünschten Containers gestartet werden. Für Docker wäre das der oben angegebene „docker run"-Befehl.
- Besteht kein Zugriff auf die Knoten, kann ein Zugriff auf die Container-Runtime über einen weiteren Pod erfolgen. Wer Docker als Container-Runtime verwendet, kann über einen Mount des Docker-Sockets weitere Container starten. Es gibt auch ein 3rd-Party-Tool, welches dies automatisiert [aylei-kubectl-debug]. Dies ist jedoch nur möglich, wenn keine PodSecurityPolicy die Ausführung von Containern mit dem User „root" und das Mounten des Docker-Sockets verhindert.
- Außerdem kann ein temporärer Pod im Network-Namespace des Hosts, also dem des Kubernetes-Knotens, starten. Von hier kann beispielsweise auf alle Interfaces des Knotens zugegriffen sowie Pakete von innerhalb des Clusters, aber außerhalb des Pod-Netzwerks versendet werden. Auch dies ist nur möglich, wenn es nicht durch eine PodSecurityPolicy verhindert wird.

Fallstricke und Tipps bei der Verwendung von Network Policies

Die Lernkurve von Network Policies ist nicht gerade flach. Daher lauern hier auch einige Fallstricke, die teilweise schon erwähnt wurden. Der folgende Abschnitt fasst diese zusammen und gibt weitere Tipps.

- Network Policies werden vom CNI-Plug-in durchgesetzt. Spezifikation und Implementierung können demnach abweichen. Also ist es empfehlenswert, am laufenden Kubernetes-Cluster zu validieren, dass die Network Policies auch gemäß Spezifikation durchgesetzt werden.
- Wenn ein CNI-Plug-in verwendet wird, das keine Network Policies unterstützt, werden vorhandene Network Policies nicht durchgesetzt. Dies kann dann zu einem falschen Gefühl der Sicherheit führen, wie es auch im Kubernetes Security Audit angemahnt wird [k8s-sec-audit].
- Für das Selektieren von Namespaces müssen den Namespaces Labels hinzugefügt werden.
- Beim Whitelisting des „ingress" besteht die Gefahr, Monitoring-Tools, Ingress-Controller und DNS zu vergessen.
- Beim Whitelisting des „egress" kann leicht die Verbindung zum Kubernetes-API-Server vergessen werden. Außerdem ist „egress" im Namespace-API generell erst ab Kubernetes Version 1.8 verfügbar.
- Die gleichzeitige Verwendung von „namespaceSelector" und „podSelector" ist erst ab Kubernetes Version 1.11 möglich.
- Nach Änderung der Network Policies ist ein Neustart aller potenziell betroffenen Pods empfehlenswert. Beispielsweise kann Prometheus nach Änderung von Network Policies teilweise zunächst noch weiter Metriken abholen. Der Fehler zeigt sich erst nach einem Neustart. Bei Traefik bleibt die Verbindung zum API-Server nach Änderung der Network Policy noch offen, beim Neustart tritt aber direkt ein Fehler auf. Bei lang laufenden Anwendungen kann es überraschend sein, dass diese beim nächsten Neustart

ohne naheliegende Änderung nicht mehr wie erwartet funktionieren. Daher ist ein Neustart direkt nach dem Ändern der Network Policies ratsam, beispielsweise mittels „kubectl rollout restart deployment" (seit „kubectl" 1.15 verfügbar).

Einschränkungen und Alternativen/ Erweiterungen

In dieser Artikelserie werden Möglichkeiten aufgezeigt, wie Network Policies zur Absicherung eines Kubernetes-Clusters beitragen können. Trotzdem gibt es einige Anforderungen, die nicht mit Network Policies realisierbar sind. Beispielsweise

- gibt es keine Möglichkeit, Cluster-weite Policies durchzusetzen,
- „egress" auf Ebene von Domain-Namen zuzulassen oder
- auf ISO/OSI-Ebene 7 (wie HTTP oder gRPC) zu filtern.

Für die Realisierung dieser Anforderungen bieten sich zwei Möglichkeiten: Nutzung proprietärer Erweiterungen der CNI-Plug-ins (Cilium und Calico bieten beispielsweise die oben genannten Möglichkeiten) oder Einsatz eines Service-Meshs wie Istio oder Linkerd.

Die Hürde für die Verwendung proprietärer Erweiterungen der CNI-Plug-ins ist relativ gering. Diese stehen als Custom Resource Definition, also als Erweiterung des Kubernetes-API zur Verfügung und können einfach in gewohnter YAML-Syntax auf den Cluster angewendet werden. Vorsicht ist allerdings beim Mischen mit Standard Network Policies geboten. Cilium rät beispielsweise hiervon ab [cilium-netpol]. Durch die Verwendung von proprietären Network Policies findet außerdem eine engere Kopplung mit dem jeweiligen CNI-Plug-in statt. Dieses kann dann schwerer ausgetauscht werden, falls beispielsweise später bessere Performanz oder andere Erweiterungen (wie Verschlüsselung) gewünscht werden.

Ein Service-Mesh bietet neben Policies auf ISO/OSI-Schicht 7 viele weitere Funktionen wie Resiliency-Patterns, Ende-zu-Ende-Verschlüsselung und Observability, die über den Kontext dieses Artikels hinausgehen. Die Verwendung von Service-Meshs erhöht allerdings die Gesamtkomplexität der Infrastruktur enorm. Wer also nur auf HTTP-Ebene filtern will, kommt mit den proprietären Features eines CNI-Plug-ins mit weniger Aufwand zum Ziel. Wer hingegen ohnehin ein Service-Mesh betreibt, kann durch die Kombination von Kubernetes Network Policies und Service-Mesh noch weitere Absicherung erreichen [isto-netpol].

```
# Temporären Pod mit bestimmten Labels starten
$ kubectl run --generator=run-pod/v1 tmp-shell --rm -i --tty \
  --labels "app=a" -n team-a --image nicolaka/netshoot -- /bin/bash

# Temporären Pod im Network Namespace des Hosts (Kubernetes node)
# starten
$ kubectl run tmp-shell --generator=run-pod/v1 --rm -i --tty \
  --overrides='{"spec": {"hostNetwork": true}}' -n team-a \
--image nicolaka/netshoot -- /bin/bash

# Debug Container in Deployment einfügen und in einem zugehörigen Pod
# eine Shell öffnen
$ kubectl patch deployment app-a -n team-a -p "$(cat <<EOF
spec:T
  template:
    spec:
    containers:
    - image: nicolaka/netshoot
    name: netshoot
    args:
    - sleep
    - '9999999999999999999'
EOF
)"
$ kubectl exec -ti $(kubectl get pod -l app=a -o jsonpath="{.items[0].
metadata.name}") \
-c netshoot bash
```

Listing 1: Netshoot-Container in Kubernetes starten

Fazit

Die Lernkurve des Kubernetes-Bordmittels Network Policies ist nicht flach. Dafür ist ihr Nutzen für die Sicherheit von auf Kubernetes-Clustern laufenden Anwendungen hoch. Der erste Artikel in dieser Serie zeigte einen pragmatischen Nutzungsweg, wie sich der Aufwand in Grenzen halten sollte. Dieser zweite Teil trägt zur Begrenzung des Aufwands bei, indem Fallstricke umgangen und Tipps zum Debugging gegeben werden. Er zeigt auf, dass es generell empfehlenswert ist, Network Policies im Cluster zu testen.

Wer darüber nachdenkt, ein Service-Mesh einzuführen, verschwendet mit Network Policies nicht seine Zeit, denn dessen Vorteile können ergänzend eingesetzt werden.

Literatur und Links

[aks-netpol] User-defined network policy in Azure Kubernetes Service (AKS) is now available | Azure updates | Microsoft Azure, https://azure.microsoft.com/en-us/updates/user-defined-network-policy-in-azure-kubernetes-service-aks-is-now-available/

[aylei-kubectl-debug] aylei/kubectl-debug: Debug your pod by a new container with every troubleshooting tools pre-installed, https://github.com/aylei/kubectl-debug

[cilium-netpol] Cilium Network Policy — Cilium 1.5.5 documentation, https://docs.cilium.io/en/v1.5/kubernetes/policy/

[eks-netpol] Tigera Calico Policy Generally Available on Amazon EKS | Tigera, https://www.tigera.io/blog/tigera-calico-policy-generally-available-on-amazon-eks/

[gke-netpol] Network policies for Kubernetes are generally available | Google Cloud Blog, https://cloudplatform.googleblog.com/2018/03/Kubernetes-Engine-network-policy-is-now-generally-available.html

[isto-netpol] Using Network Policy with Istio, https://istio.io/blog/2017/0.1-using-network-policy/

[k8s-sec-audit] Open Sourcing the Kubernetes Security Audit - Cloud Native Computing Foundation, https://www.cncf.io/blog/2019/08/06/open-sourcing-the-kubernetes-security-audit/

[k8s-sec-demo] GitHub: cloudogu/k8s-security-demos, https://github.com/cloudogu/k8s-security-demos/tree/master/2-network-policies

[kubectl-debug] kubernetes/community: Troubleshoot Running Pods, https://github.com/kubernetes/community/blob/master/contributors/design-proposals/node/troubleshoot-running-pods.md

[netpol-e2e-test] Testing Kubernetes Network Policy Enforcement with Sonobuoy, alexbrand's blog, https://alexbrand.dev/post/testing-kubernetes-network-policy-enforcement-with-sonobuoy/

[Sch19] J. Schnatterer, Network Policies: Teil 1 - Good Practices, in: JavaSPEKTRUM, 05/2019, https://cloudogu.com/de/blog/k8s-app-ops-teil-1

[test-netpol] Why You Should Test Your Kubernetes Network Policies - inovex-Blog, https://www.inovex.de/blog/test-kubernetes-network-policies/